Kauno technologijos universitetas

Informatikos fakultetas

Laboratorinių darbų ataskaita

P175B124 Programavimo kalbų teorija

Autorius
Gustas Klevinskas

Akademinė grupė
IFF-8/7

Vadovai
Evaldas Guogis
Svajūnas Sajavičius

Kaunas, 2021

Turinys

1 Python (L1)

1.1 Darbo užduotis

„Minesweeper“ žaidimas. Nuoroda į užduotį: https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=29&page=show_problem&problem=1130.

Duomenys sudaryti iš neapibrėžto kiekio laukų. Kiekvieno lauko pirmojoje eilutėje pateikti su sveikieji skaičiai n ir m ($0 < n, m \leq 100$), kurie atitinkamai pažymi eilučių ir stulpelių skaičių lauke. Laukas sudarytas iš dviejų simbolių: žvaigždutes ir taško („*“ ir „.“). Duomenų pateikimo stabdymas nurodomas pateikiant $n = m = 0$ reikšmes.

Kiekvienam laukui reikia atspausdinti žinutę „Field #x“, kur x – lauko numeris (pradedant nuo 1). Sekančiose n eilučių pateikti lauką, kuriame taško simboliai pakeisti į skaičius, nurodančius aplink tą tašką esančių žvaigždučių kiekį. Tarp laukų turi būti tuščia eilutė.

1 lentelė. *Pavyzdiniai duomenys ir rezultatai*

Pavyzdiniai duomenys	Rezultatai
4 4 *... .... .*.. .... 3 5 **... ..... .*... 0 0	Field #1: *100 2210 1*10 1110 Field #2: **100 33200 1*100

1.2 Programos tekstas

1.2.1 main.py

```
# Programming Challenges (Skiena & Revilla)
# Chapter 1
# 10189 - Minesweeper

import re
from field import Field

if __name__ == '__main__':
    input_file = open("input.txt", "r")
    next(input_file)

    field = Field()
    i = 1

    for line in input_file:
        if re.search(r"\d \d", line) is not None:
            field.print_board(i)

            field = Field()
            i += 1
            continue

        # Remove \n from the end
        field.set_row(line[:-1])
```

1.2.2 field.py

```
class Field:

    def __init__(self):
        self.field = []

    def set_row(self, line):
        row = []

        for i in line:
            if i == ".":
                row.append(0)
            else:
                row.append(i)

        self.field.append(row)

    def solve(self):
        for y, row in enumerate(self.field):
            for x, cell in enumerate(row):
                if cell == "*":
                    self.set_mine(x, y)

    def set_mine(self, x, y):
        # Row 1
        self.increment(x - 1, y - 1)
        self.increment(x, y - 1)
        self.increment(x + 1, y - 1)

        # Row 2
        self.increment(x - 1, y)
        self.increment(x + 1, y)

        # Row 3
        self.increment(x - 1, y + 1)
        self.increment(x, y + 1)
        self.increment(x + 1, y + 1)

    def increment(self, x, y):
        if self.bounds_ok(x, y):
            self.field[y][x] += 1

    def bounds_ok(self, x, y):
        return (0 <= y < len(self.field) and
                0 <= x < len(self.field[y]) and
```

```
            type(self.field[y][x]) is int)

    def print_board(self, i):
        self.solve()

        print("Field #{}:".format(i))
        for row in self.field:
            print(''.join(map(str, row)))
        print("")
```

1.3 Pradiniai duomenys ir rezultatai

2 lentelė. *Programai pateikti duomenys ir iš jos gauti rezultatai*

Duomenys	Rezultatai
4 4 *... .... .*.. .... 3 5 **... ..... .*... 1 1 * 1 1 . 1 4 .*.. 0 0	Field #1: *100 2210 1*10 1110 Field #2: **100 33200 1*100 Field #3: * Field #4: 0 Field #5: 1*10

2 Scala (L2)

2.1 Darbo užduotis

Naudojant įrankį „Scalatron“, sukurti robotą. Robotas turi paleisti padėjėjus robotukus (minos, raketos, rinkikai ir pan.) bei naudoti kelio paieškos algoritmą.

Pasirinkti realizuoti robotukai:

- Puolikas – juda link artimiausio priešo ir sprogsta.
- Bomba – priartėjus priešui robotukas sprogsta.
- Raketa – tiesia trajektorija paleidžiamas robotukas, kuris irgi sprogsta priartėjus priešui.

Kelio paieškos algoritmas realizuotas A* algoritmu.

2.2 Programos tekstas

```
import util.Random
import scala.collection.mutable.Queue

class ControlFunctionFactory {
    def create = new ControlFunction().respond _
}

class ControlFunction {
    val rand = new Random()
    var currentRoute = List[XY]()

    def respond(input: String): String = {
        val (operation, params) = CommandParser(input)

        if (operation == "React") {
            var generation = params("generation").toInt
            var energy = params("energy").toInt
            var view = View(params("view"))

            if (generation == 0) {
                if (energy >= 100 && rand.nextDouble() < 0.1 && (view.cells.contains('b') ||
view.cells.contains('m'))) {
                    def heading = XY.random(rand)
                    "Spawn(bottype=attacker,direction=" + heading + ",energy=100,heading=" + heading + ")"
                } else if (energy >= 400 && rand.nextDouble() < 0.05) {
                    "Spawn(bottype=mine,direction=" + XY.random(rand) + ",energy=200)"
                } else if (energy >= 100 && rand.nextDouble() < 0.15 && (view.cells.contains('b') ||
view.cells.contains('m'))) {
                    view.offsetToNearestEnemy match {
                        case Some(offset) => "Spawn(bottype=rocket,direction=" + offset.signum +
",energy=100,heading=" + offset.signum + ")"
                        case None => ""
                    }
                } else {
                    if (view.cells contains "P") {
                        routeMove(view)
                    } else {
                        view.offsetToNearestEnemy match {
                            case Some(enemyOffset) =>
                                if (enemyOffset.length < 8) {
                                    currentRoute = List[XY]()
                                    "Move(direction=" + view.safeMove(enemyOffset.inverted, rand) + ")"
                                } else {
                                    "Move(direction=" + view.safeMove(XY.random(rand), rand) + ")"
                                }
                            case None => "Move(direction=" + view.safeMove(XY.random(rand), rand) + ")"
                        }
                    }
                }
            } else { // Slave
                val botType = params("bottype")

                if (botType == "attacker") {
                    view.offsetToNearestEnemy match {
                        case Some(offset) =>
                            if (offset.length < 2) {
                                "Explode(size=2)"
                            } else {
                                val enemyOffset = offset.signum
                                "Move(direction=" + enemyOffset + ")"
                            }
                        case None =>
                            val heading = XY(params("heading"))
                            "Move(direction=" + view.safeMove(heading, rand) + ")"
                    }
                } else if (botType == "mine") {
                    view.offsetToNearestEnemy match {
                        case Some(offsetEnemy) =>
                            if (offsetEnemy.length < 6) {
                                view.offsetToNearest('M') match {
                                    case Some(offsetMaster) => {
                                        if (offsetMaster.length > 6) {
                                            "Explode(size=8)"
                                        } else ""
```

```
                                }
                                case None => "Explode(size=8)"
                            }
                        } else ""
                    case None => ""
                }
            } else if (botType == "rocket") {
                view.offsetToNearestEnemy match {
                    case Some(offsetEnemy) =>
                        if (offsetEnemy.length < 4) {
                            "Explode(size=6)"
                        } else "Move(direction=" + params("heading") + ")"
                    case None => "Move(direction=" + params("heading") + ")"
                }
            } else ""
        }
    } else ""
  }

  def routeMove(view: View) = {
    if (currentRoute.length == 0) {
      currentRoute = view.astar
    }

    val direction = currentRoute.head
    currentRoute = currentRoute.drop(1)
    currentRoute = currentRoute.map(_ - direction)
    "Move(direction=" + direction + ")"
  }
}


case class XY(val xcoord: Int, val ycoord: Int) {
  override def toString = xcoord + ":" + ycoord
  def +(other: XY) = new XY(xcoord+other.xcoord, ycoord+other.ycoord)
  def -(other: XY) = new XY(xcoord-other.xcoord, ycoord-other.ycoord)
  def inverted = XY(-xcoord, -ycoord)
  def signum = XY(xcoord.signum, ycoord.signum)
  def distanceTo(pos: XY): Double = (this - pos).length
  def length: Double = math.sqrt(xcoord*xcoord + ycoord*ycoord)
  def squaredLength = xcoord * xcoord + ycoord * ycoord
  def squaredDistanceTo(pos: XY): Int = (this - pos).squaredLength
}


object XY {
  def random(rnd: Random) = XY(rnd.nextInt(3) - 1, rnd.nextInt(3) - 1)

  def apply(str: String): XY = {
    val c = str.split(':').map(_.toInt)
    XY(c(0), c(1))
  }

  val right = XY(1, 0)
  val rightUp = XY(1, -1)
  val up = XY(0, -1)
  val upLeft = XY(-1, -1)
  val left = XY(-1, 0)
  val leftDown = XY(-1, 1)
  val down = XY(0, 1)
  val downRight = XY(1, 1)
}


case class View(cells: String) {
  val size = math.sqrt(cells.length).intValue
  val center = XY(size / 2, size / 2)

  def apply(absPos: XY) = cells.charAt(absPosToIndex(absPos))

  def absPosToIndex(absPos: XY) = absPos.xcoord + absPos.ycoord * size
  def indexToAbsPos(index: Int) = XY(index % size, index / size)
  def relPosToAbsPos(relPos: XY) = relPos + center
  def cellAtAbsPos(absPos: XY) = cells(absPosToIndex(absPos))

  def relPosToIndex(relPos: XY) = absPosToIndex(relPosToAbsPos(relPos))
```

```
def absPosToRelPos(absPos: XY) = absPos - center
def indexToRelPos(index: Int) = absPosToRelPos(indexToAbsPos(index))
def cellAtRelPos(relPos: XY) = cells(relPosToIndex(relPos))

def isSafe(c: Char) = {
    !(c == 'W' || c == 'p' || c == 'b' || c == 'm' || c == 's')
}

def safeMove(move: XY, rand: Random): XY = {
    val dest = apply(relPosToAbsPos(move))

    if (!isSafe(dest)) {
        val inverted = move.inverted
        if (isSafe(apply(relPosToAbsPos(inverted)))) {
            inverted
        } else safeMove(XY.random(rand), rand)
    } else move
}

def offsetToNearestEnemy(): Option[XY] = {
    offsetToNearest('m') match {
        case Some(offsetm) =>
            offsetToNearest('b') match {
                case Some(offsetb) =>
                    if (offsetb.length < offsetm.length)
                        Some(offsetb)
                    else Some(offsetm)
                case None => Some(offsetm)
            }
        case None =>
            offsetToNearest('b') match {
                case Some(offsetb) => Some(offsetb)
                case None => None
            }
    }
}

def offsetToNearest(c: Char): Option[XY] = {
    val relativePositions = cells
      .view
      .zipWithIndex
      .filter(x => x._1 == c)
      .map(x => indexToRelPos(x._2))

    if (!relativePositions.isEmpty)
        Some(relativePositions.minBy(x => x.length))
    else None
}

def astar = {
    def visitAstarNode(xy: XY, visitor: AstarNode, nodes: List[AstarNode]) = {
        if (apply(relPosToAbsPos(xy)) != 'W' && apply(relPosToAbsPos(xy)) != 'p') {
            if (!nodes.map(_.xy).contains(xy)) {
                var newNode = AstarNode(xy, apply(relPosToAbsPos(xy)), visitor.g + 1)
                newNode.previous = Some(visitor)
                nodes :+ newNode
            } else {
                var existingNode = nodes.find(x => xy == x.xy).get
                if (existingNode.g > visitor.g + 1) {
                    var newNode = AstarNode(xy, apply(relPosToAbsPos(xy)), visitor.g + 1)
                    existingNode.previous = Some(visitor)
                    nodes.map(x => if (x.xy == existingNode.xy) newNode else x)
                } else nodes
            }
        } else nodes
    }

    var nodes = List[AstarNode]()
    var start = AstarNode(XY(0,0), 'M', 0)
    var end = AstarNode(offsetToNearest('P').get, 'P', 0)
    var route = List[XY]()
    nodes = nodes :+ start

    while (nodes.filter(x => x.isOpen).length > 0) {
        var lowestNode = nodes.filter(x => x.isOpen).minBy(x => x.getF(end.xy))
        lowestNode.isOpen = false
```

```
            if (lowestNode.chr == 'P') {
                var node = lowestNode
                while (node.previous.isDefined) {
                    route = route :+ node.xy
                    node = node.previous.get
                }
                nodes = List[AstarNode]()
            } else {
                nodes = nodes.map(x => if (x.xy == lowestNode.xy) lowestNode else x)
                nodes = visitAstarNode(lowestNode.xy + XY.right, lowestNode, nodes)
                nodes = visitAstarNode(lowestNode.xy + XY.rightUp, lowestNode, nodes)
                nodes = visitAstarNode(lowestNode.xy + XY.up, lowestNode, nodes)
                nodes = visitAstarNode(lowestNode.xy + XY.upLeft, lowestNode, nodes)
                nodes = visitAstarNode(lowestNode.xy + XY.left, lowestNode, nodes)
                nodes = visitAstarNode(lowestNode.xy + XY.leftDown, lowestNode, nodes)
                nodes = visitAstarNode(lowestNode.xy + XY.down, lowestNode, nodes)
                nodes = visitAstarNode(lowestNode.xy + XY.downRight, lowestNode, nodes)
            }
        }

        route.reverse
    }
}


case class AstarNode(xy: XY, chr: Char, g: Int) {
    var previous: Option[AstarNode] = None
    var isOpen = true
    def getH(finish: XY): Int = xy.squaredDistanceTo(finish)
    def getF(finish: XY): Int = if (g >= 0) getH(finish) + g else -1
}


object CommandParser {
    def apply(command: String) = {
        val tokens = command.split('(')
        val paramMap = tokens(1).dropRight(1)
          .split(',')
          .map(s => s.split('='))
          .map(arr => (arr(0), arr(1)))
          .toMap

        (tokens(0), paramMap)
    }
}
```

2.3 Rezultatai

Robotą sukurti pavyko, tačiau dar toli iki efektyvaus veikimo. Gana dažnai atsitinka situacija, kai jį apsupa priešiški monstrai, tuomet robotui sunku išsprūsti iš jų, o apsiginti irgi negali, nes nebelieka energijos puolikų robotukų kūrimui.

Šia bėdą būtų galima išspręsti A* algoritme atsižvelgus į priešų būvimo vietą, tačiau tai ženkliai apsunkintų algoritmą. Šiuo metu dėl paprastumo algoritmas ieško kelio iki artimiausio valgomo augalo.

3 Haskell (L3)

3.1 Darbo užduotis

Atlikti skaičiavimus su duotais skaičiais. Nuoroda į užduotį: https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=121&page=show_problem&problem=2542.

Pirmoje duomenų eilutėje pateiktas testinių atvejų skaičius (kiek bus tolimesnių eilučių, su kuriomis reikės atlikti skaičiavimus). Sekančiose eilutėse pateikti skaičiai. Reikia grąžinti prieš paskutinį skaitmenį pritaikius (1) formulę, kur *n* – duotas skaičius.

$$\Big(\big((n \cdot 567) \div 9\big) + 7492\Big) \cdot 235\Big) \div 47 - 498 \quad (1)$$

Supaprastinus daugybą ir dalybą gaunama (2) formulė:

$$\big((n \cdot 63) + 7492\big) \cdot 5 - 498 \quad (2)$$

3 lentelė. *Pavyzdiniai duomenys ir rezultatai*

Pavyzdiniai duomenys	Rezultatai
2 637 -120	1 3

3.2 Programos tekstas

```
fun :: String -> Integer
fun x = abs $ (((read x * 63) + 7492) * 5) - 498

tens :: Integer -> Integer
tens x = (x `div` 10) `mod` 10

calcAndPrint :: [String] -> IO ()
calcAndPrint [] = return ()
calcAndPrint (x:xs) = do
    putStrLn $ show $ tens (fun x)
    calcAndPrint xs

main :: IO ()
main = do
    content <- readFile "input.txt"
    calcAndPrint (tail (lines content))
```

3.3 Pradiniai duomenys ir rezultatai

4 lentelėje pateikti parašytos programos rezultatai su pavyzdiniais duomenimis bei naujais testiniais variantais.

4 lentelė. *Programai pateikti duomenys ir gauti rezultatai*

Pavyzdiniai duomenys	Rezultatai
4 637 -120 1000 0 -1000	1 3 6 6 3